CPSC 314
Assignment 3: Lighting and Shading

Due 11:59PM, Feb 27 2019

1 Introduction

The goal of this assignment is to explore lighting and shading of 3D models. You will be simulating five different models: Phong, Blinn-Phong, Spotlight, Fog(the closer, the 'clearer'), Toon (cartoon-like). In this assignment you will be *completing* the five supplied pairs of shaders (both vertex and fragment shaders) to carry out these tasks, which includes editing necessary parts of the A3.js file.

1.1 Getting the Code

Assignment code is hosted on UBC STASH BitBucket, to retrieve it onto your local machine first ensure that you have logged in at least once onto BitBucket:

`https://stash.ugrad.cs.ubc.ca:8443`.

Then navigate to the folder on your machine where you intend to keep your assignment code, and run the following command from the terminal or command line:

```
git clone https://stash.ugrad.cs.ubc.ca:8443/scm/cpsc314-2018wt2_students/a3.git
```

1.2 Template

There are five scenes in the assignment, accessible with the number keys 0, 1, 2, 3, 4 and 5 on your keyboard. You will be implementing the lighting models Phong in Scene 1, Blinn-Phong in Scene 2, Fog in Scene 3, Spotlight in Scene 4 and Toon in Scene 5.
The template code is found in the main assignment directory, shader files can be found in the glsl folder, and the obj folder contains different objects at your disposal for the last scene.
We have already defined the properties of a basic directional light in A3.js for the scene (defined in the lightColor, lightFogColor, lightDirection, spotlightPosition and ambientColor variables). We have also defined suggested material properties for the objects you will be shading (defined in kSpecular, kDiffuse, kAmbient, shininess and fogDensity).

1.3 Important Rules

The lighting models you will be implementing in this assignment are very common in graphics applications. Thus three.js has already implemented them in a number of the default materials provided with three.js. These can be used to test if you are getting reasonable results, but you must implement these shading models yourself in your own custom shaders.

2 Work to be done (100 pts)

1. **Part 1:** (70 pts) Shaders.

   (a) **15 pts** Scene 1: Phong Reflection and Phong Shading. In Scene 1-3 you will see many eggs. Your goal is to shade them, using Gouraud, Phong, Blinn-Phong and Fog models in different scenes. The Gouraud shaded sphere has already been given to you in Scene 0 for visual comparison with the other shading models, so you don't need to implement it.
   The Phong *shading* model improves on the Gouraud shader by computing the lighting per fragment, rather than per vertex. This is done by using the interpolated values of the fragment's position and normal.

   The following image, taken from the Wikipedia article on the Phong reflection model, shows how the different components look individually and summed together:

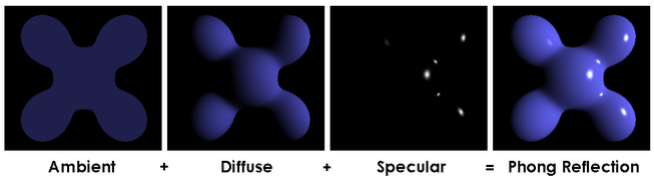

Figure 1: Phong Reflection Model

   For this part implement the full Phong reflection model in a Phong shader, and apply it to the eggs in Scene 1. The main calculations should all go in the fragment shader (in file `phong.fs.glsl`). Note that you will still need a vertex shader (in `phong.vs.glsl`), to pass the appropriate information to your fragment shader. Remember that dot products should be clamped between 0 and 1, since we don't want the objects to be backlit. Your result should look similar to Figure 2.

   Hint: `normalMatrix` helps to take `normal` into eye space.

   (b) **10 pts** Scene 2: Blinn-Phong Reflection.

   Instead of continuously recomputing the dot product between the reflected light vector and viewer, a dot product between the halfway vector between light and viewing direction, and the surface normal can be used. This is the essence of the Blinn-Phong reflection model.

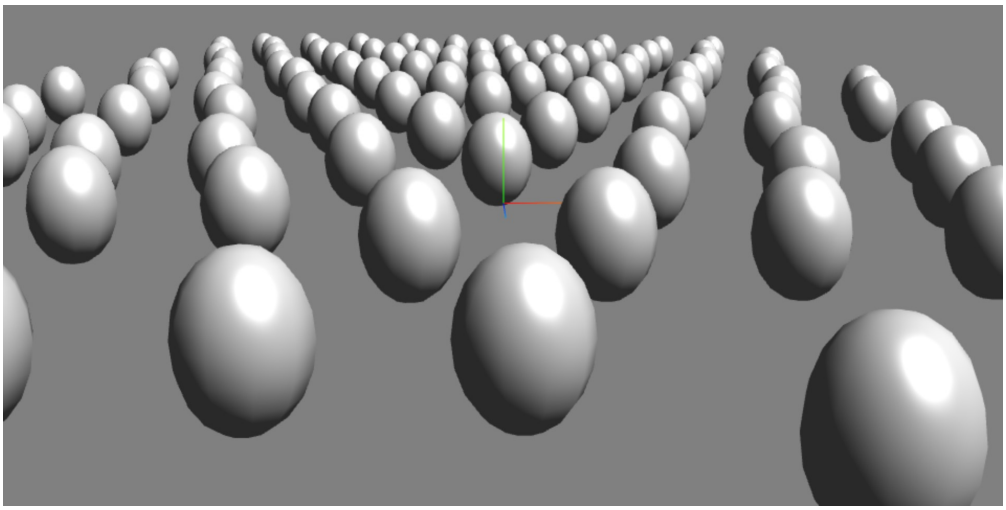

Figure 2: Example of completed Scene 1.

Complete the supplied Blinn-Phong shaders (`phong_blinn.fs.glsl` and `phong_blinn.vs.glsl`) to apply the Blinn-Phong reflection model, per fragment, to the eggs in Scene 2.

Once Scene 2 is complete, it should look similar to Figure 3. You can also use the built in THREE.js Phong materials to test your shader work.

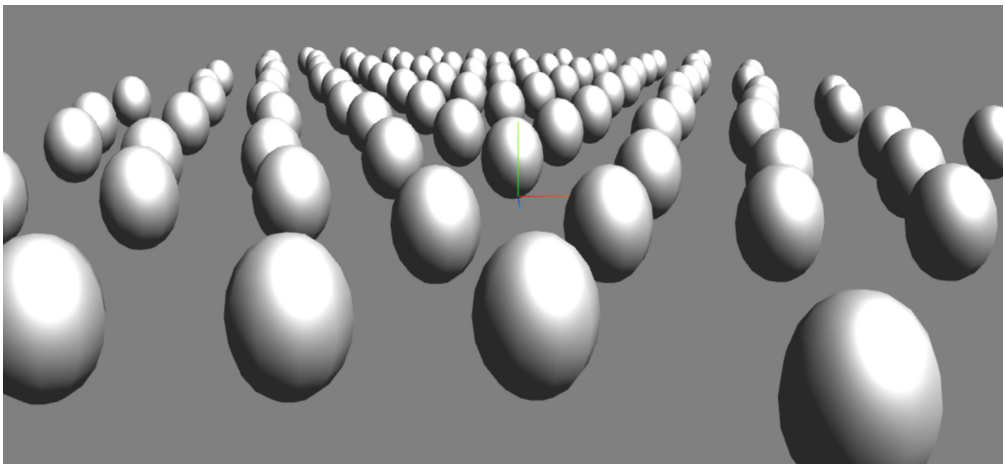

Figure 3: Example of completed Scene 2.

Note: the "Gouraud" THREE.js material does not have a specular component. *Hint:* Specular lighting shouldn't be static as you move the camera around the scene!

(c) **15 pts** Scene 3: Fog Simulation
In this part, you would modify the shader to simulate fog, i.e. the objects that are closer to the camera are clearer while the further ones are foggier. Complete the supplied Fog shaders(`fog.vs.glsl` and `fog.fs.glsl`) to build the model, per

fragment, to the eggs in Scene 3.
You should compute the distance between the vertex and the camera. Then you're required to apply exponential fog equation to get `FogLevel` for interpolation. $FogLevel = 1/e^{distance*fogDensity}$

The final fragment color will be an interpolated value between the fog color(`lightFogColor`) and the simple diffuse color(you should calculate it by yourself). You should get a similar scene as Figure 4.

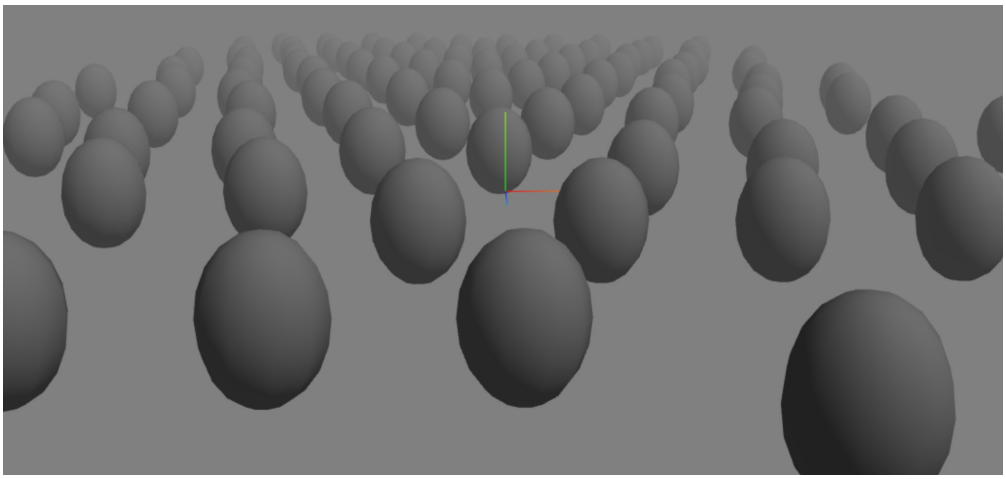

Figure 4: Example of completed Scene 3.

(d) **15 pts** Scene 4: Moving Spotlight

A spotlight is a kind of point light, which emits only a cone of light. The vertex of the cone is located at the position of the light(`spotlightPosition`). The cone points in some direction, called the spot direction, is specified as a vector. The intensity of light is dependent on the angle between the spot direction and the light ray. Your goal for this part is to create a spotlight with `spotlightPosition` fixed and spot direction moving by keyboard control. In `spotlight.fs.glsl`, you will need to judge whether the fragments can be lit by the spotlight or not using the angle(threshold set by yourself). Then the fragments will be colored with the intensity $I * c^s$. (I is the `SpotColor`, c is the cosine of the angle and s is `spotExponent`). After modifying the shaders, you will need to apply this material to bunny and floor to see how it works. Your completed scene should be similar to Figure 5.

(e) **15 pts** Scene 5: Toon Shading with silhouettes.

In Scene 5, you will start off with just a sphere for testing your implementation of a Toon shading model. This is an example of a "non-photo realistic" (or NPR) shader. It emulates the way cartoons use very few colors for shading, and the color changes abruptly, while still providing a sense of 3D for the model. This can be implemented by quantizing the light intensity across the surface of the object. Instead of making the intensity vary smoothly, you quantize this variation into a

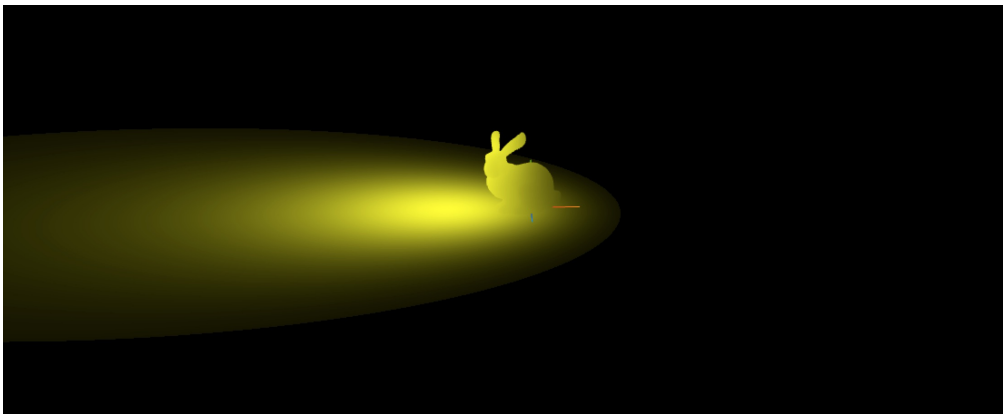

Figure 5: Example of completed Scene 4.

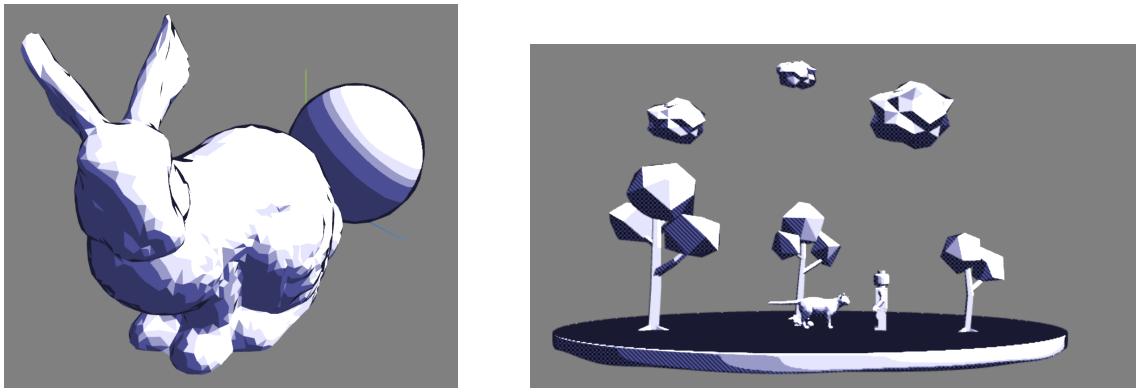

Figure 6: Example of toon with silhouettes

number of steps for each "layer" of toon shading.
Also, draw approximate silhouette edges of the object by detecting fragments on the silhouette and coloring these fragments with a dark color (think about the relationship between the surface normal and the viewing direction for these fragments).
Like before, begin by implementing the Toon shaders
(`toon.fs.glsl` and `toon.vs.glsl`) for the sphere, and once the sphere looks correct, build an interesting scene using the provided objects in the obj folder of the assignment, and the object loader in A3.js. See Figure 6.

If you need some inspiration the following movies and video games were rendered with toon shading (also called cell shading) techniques:

`http://en.wikipedia.org/wiki/List_of_cel-shaded_video_games`.

2. **Part 2:** (30 pts) Creative License. Create a new shader or modify one of the above shaders to implement some interesting lighting effects similar to the complexity of the ones above. Some recommendations are:

   - Implement cross-hatching in the toonshader

- Implement multiple light sources in one scene e.g. point light, spotlight, directional light.
- Explore other non-photo realistic shaders, such as Gooch shading for technical illustration.
- Add more complexity to the fog model.
- Implement a more physically based shader.
- Animate the objects in toon shader to create an interesting demo

Bonus marks may be given at the discretion of the marker for particularly noteworthy explorations.

2.1 Hand-in Instructions

You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and login ID, and any information you would like to pass on to the marker, including notes on your creative license(keystroke, story, etc.). Create a folder called "a3" under your "cs314" directory. Within this directory have two subdirectories named "part1," and "part2", and put all the source files, your makefile, and your README.txt file for each part in the respective folder. Do not use further sub-directories. The assignment should be handed in with the exact command:

```
handin cs314 a3
```